

# Pipeline e progetto dell'ISA

- L'ISA del MIPS è progettata per il pipelining
  - Tutte le istruzioni sono a 32 bit
    - Più facile prelevare e decodificare in un ciclo
    - c.f. x86: istruzioni da 1 a 17 bit
  - Pochi e regolari formati di istruzioni
    - Si possono decodificare e leggere i registri in un passo
  - Indirizzamento load/store
    - Si può calcolare l'indirizzo nel 3° stadio e accedere alla memoria nel 4° stadio
  - Allineamento in memoria degli operandi
    - Un accesso alla memoria richiede un solo ciclo

# Hazard (criticità)

- Situazioni che impediscono l'inizio della prossima istruzione nel prossimo ciclo
- Hazard strutturali
  - Una risorsa necessaria è occupata
- Hazard sui dati
  - Bisogna attendere che l'istruzioni completi la sua lettura/scrrittura del dato
- Hazard sul controllo
  - La decisione sull'azione di controllo dipende dall'istruzione precedente

# Hazard strutturali

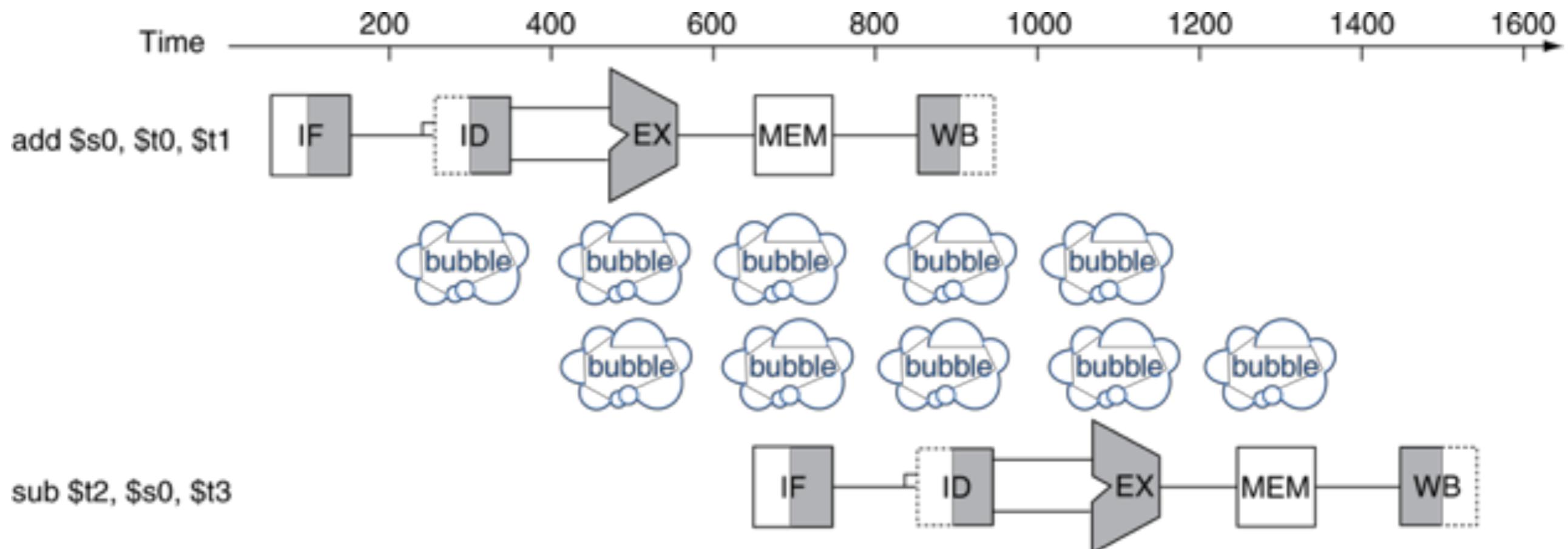
- Conflitto per l'uso di una risorsa
- Nella pipeline del MIPS con una singola memoria
  - Load/store richiede l'accesso al dato
  - Il fetch dell'istruzione dovrebbe andare in *stallo* per quel ciclo
    - Causerebbe una *bolla* della pipeline
- Quindi, le unità di elaborazione in pipeline richiedono memorie istruzioni/dati separate
  - O cache istruzioni/dati separate

# Hazard sui dati

- Un'istruzione dipende dal completamento dell'accesso ai dati di un'istruzione precedente

add \$s0, \$t0, \$t1

sub \$t2, \$s0, \$t3



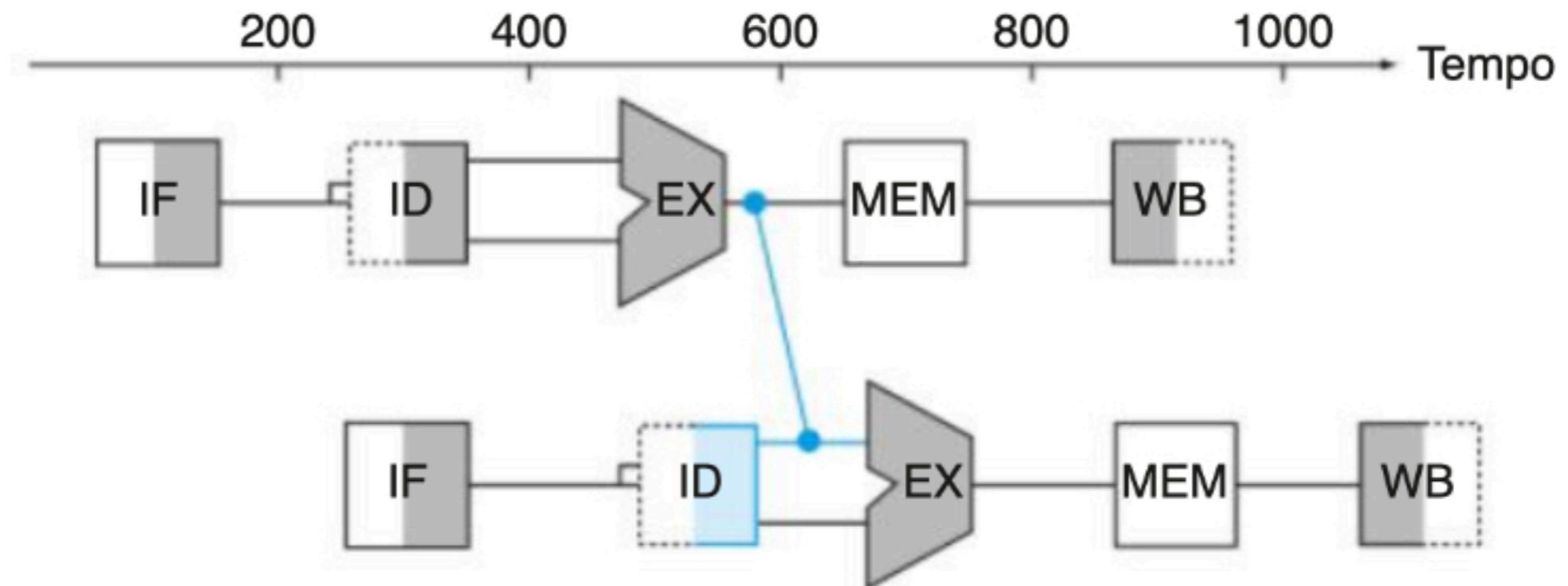
# Propagazione (bypassing)

- Usare il risultato quando è stato calcolato
  - Non aspettare che sia memorizzato in un registro
  - Richiede connessioni extra nell'unità di elaborazione

Ordine di esecuzione  
del programma  
(sequenza  
delle istruzioni)

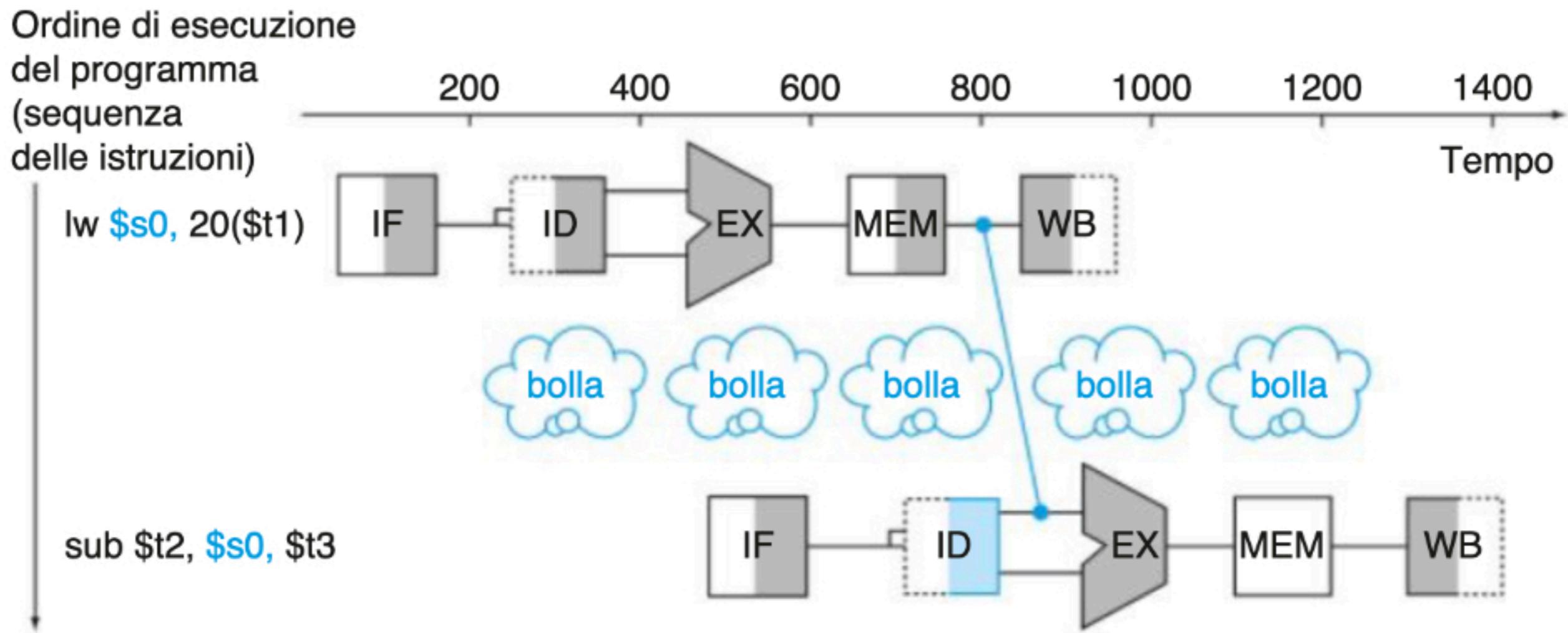
add \$s0, \$t0, \$t1

sub \$t2, \$s0, \$t3



# Hazard sui dati di una load

- Non si possono evitare sempre gli stalli tramite propagazione
  - Se il valore non è ancora calcolato quando necessario
  - Non si può propagare indietro nel tempo!

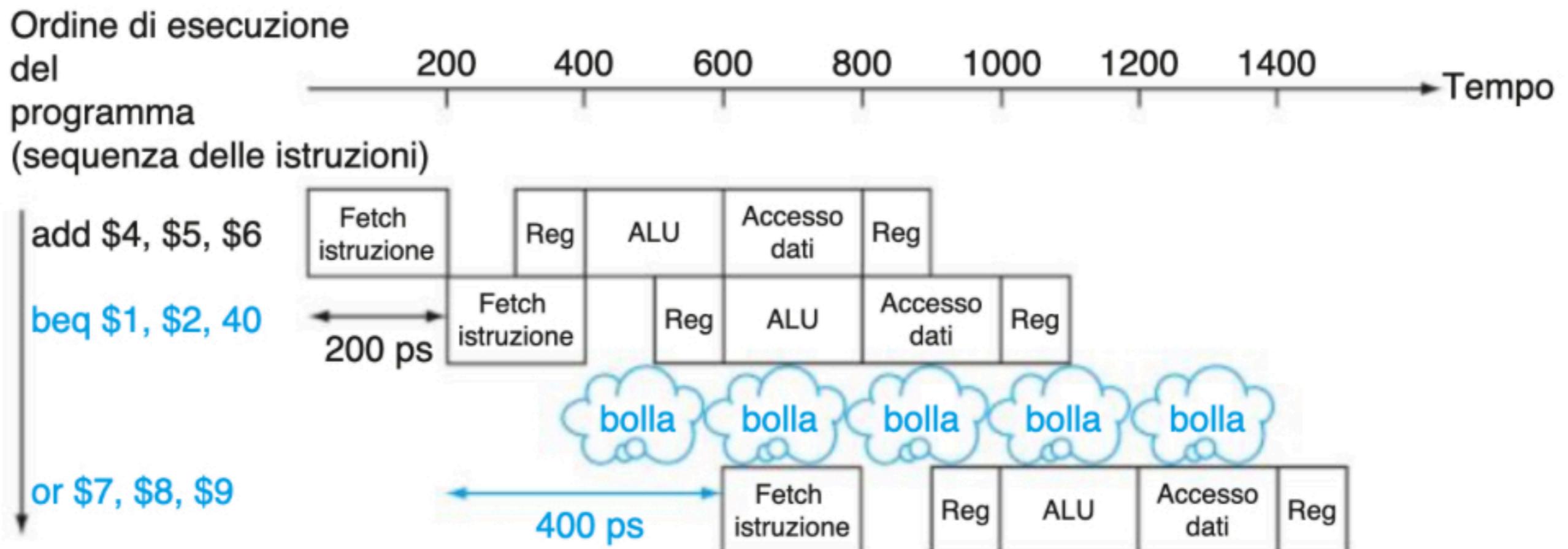


# Hazard sul controllo

- Un salto determina il flusso del controllo
  - Il prelievo della prossima istruzione dipende dal risultato del salto
  - La pipeline non può prelevare sempre l'istruzione corretta
    - Se sta ancora elaborando lo stadio ID del salto
- Nella pipeline del MIPS
  - E' necessario confrontare i registri e calcolare la destinazione prima nella pipeline
  - Aggiungere hardware per farlo nello stadio ID

# Stallo dovuto ai salti

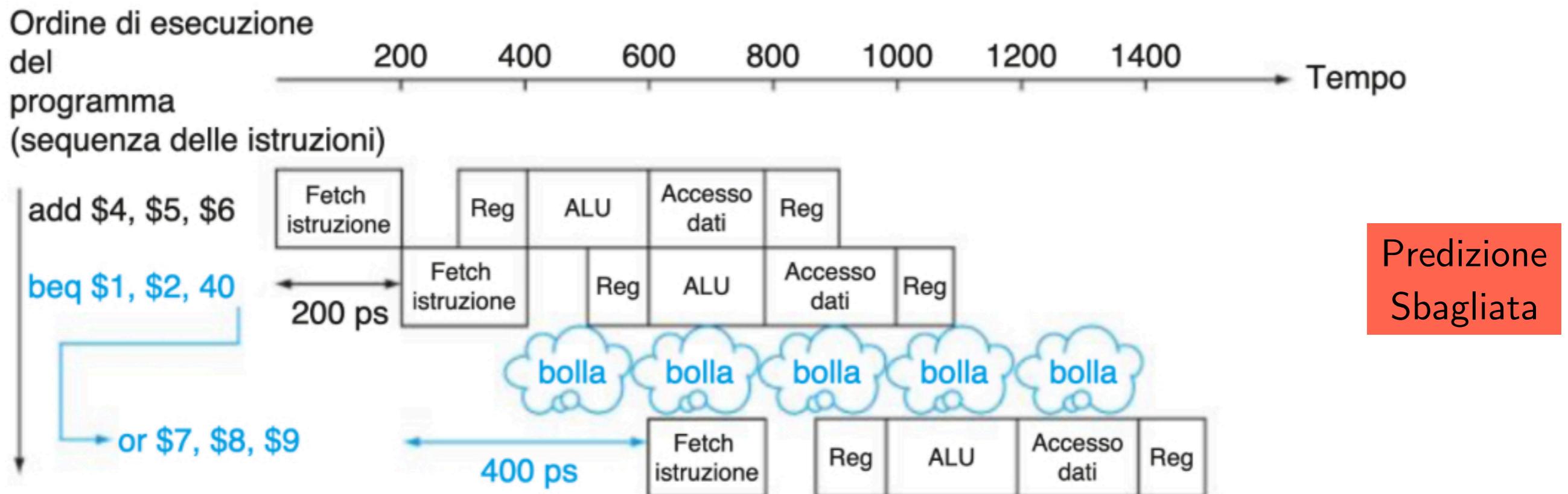
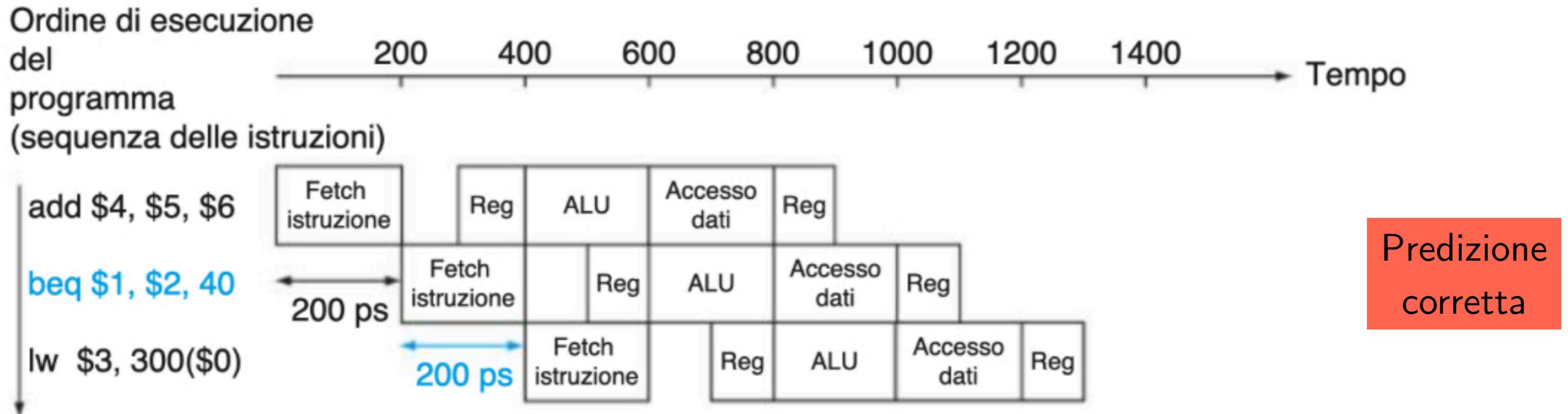
- Bisogna attendere finché il risultato del salto non è stato calcolato prima di prelevare la prossima istruzione



# Predizione dei salti

- Le pipeline più lunghe non possono determinare per tempo il risultato di un salto
  - La penalità di stallo diventa inaccettabile
- Predizione del risultato di un salto
  - Si entra in stallo solo se la predizione è sbagliata
- Nella pipeline del MIPS
  - E' possibile predire i salti non effettuati
  - Prelievo dell'istruzione dopo il salto, senza ritardo

# MIPS con predizione



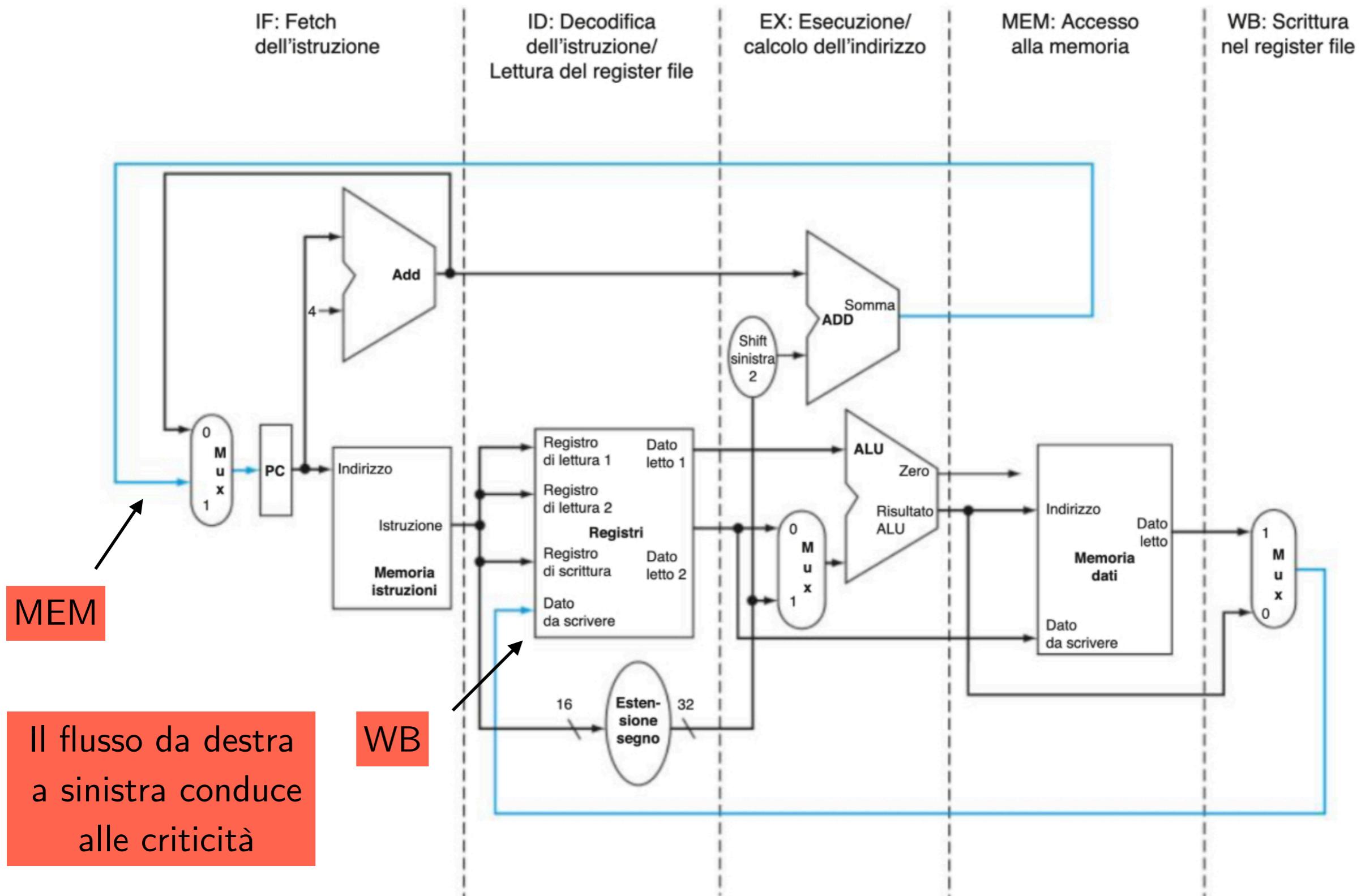
# Predizione dei salti più realistica

- Predittore statico
  - Si basa sul comportamento tipico dei salti
  - Esempio: salti nei cicli e nei comandi condizionali
    - Si predice che i salti all'indietro sono scelti
    - Si predice che i salti in avanti non sono scelti
- Predittore dinamico
  - L'hardware misura il comportamento attuale del salto
    - Per esempio, registrando la storia recente di ogni salto
  - Assume che il comportamento futuro seguirà il trend
    - Quando si sbaglia, si entra in stallo con nuovo prelievo e si aggiorna la storia

# Riepilogo sulla pipeline

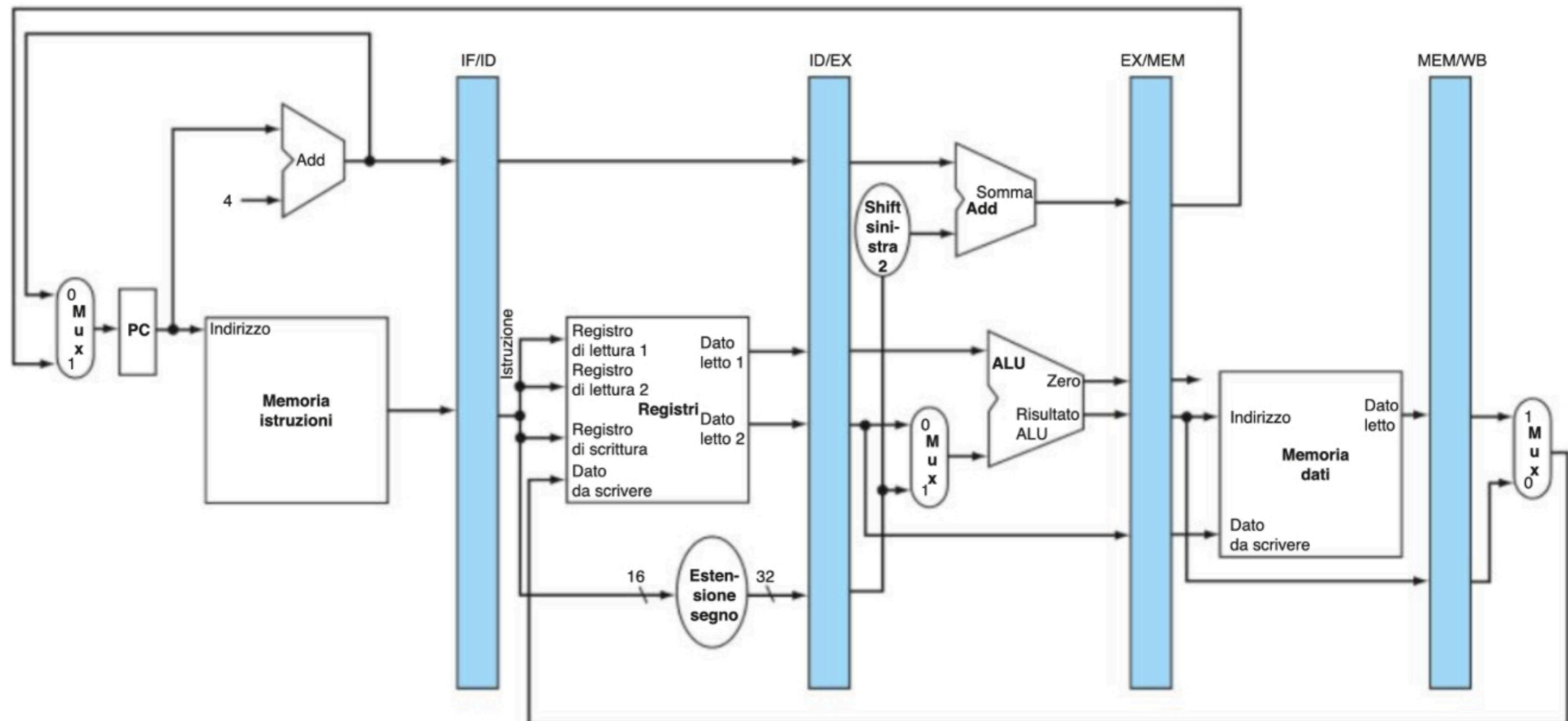
- Il pipelining migliora le prestazioni incrementando il throughput delle istruzioni
  - Esegue molteplici istruzioni in parallelo
  - Ogni istruzione ha la stessa latenza
- Soggetta a hazard
  - Strutturali, sui dati, sul controllo
- Il progetto dell'insieme di istruzioni ha un impatto sulla complessità dell'implementazione della pipeline

# Datapath con pipeline del MIPS



# Registri Pipeline

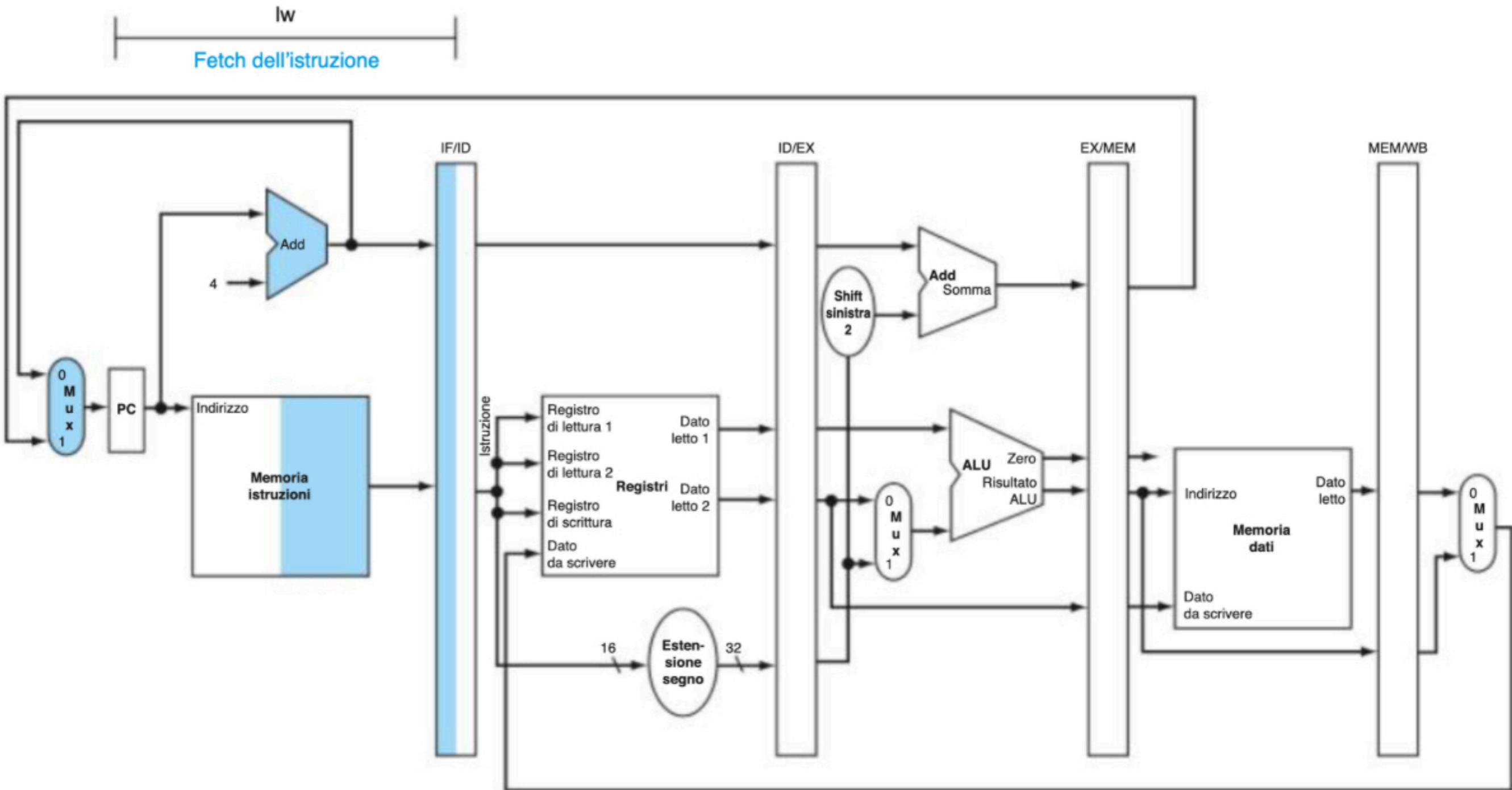
- Sono necessari dei registri tra i vari stadi
  - Per immagazzinare le informazioni prodotte nel ciclo precedente



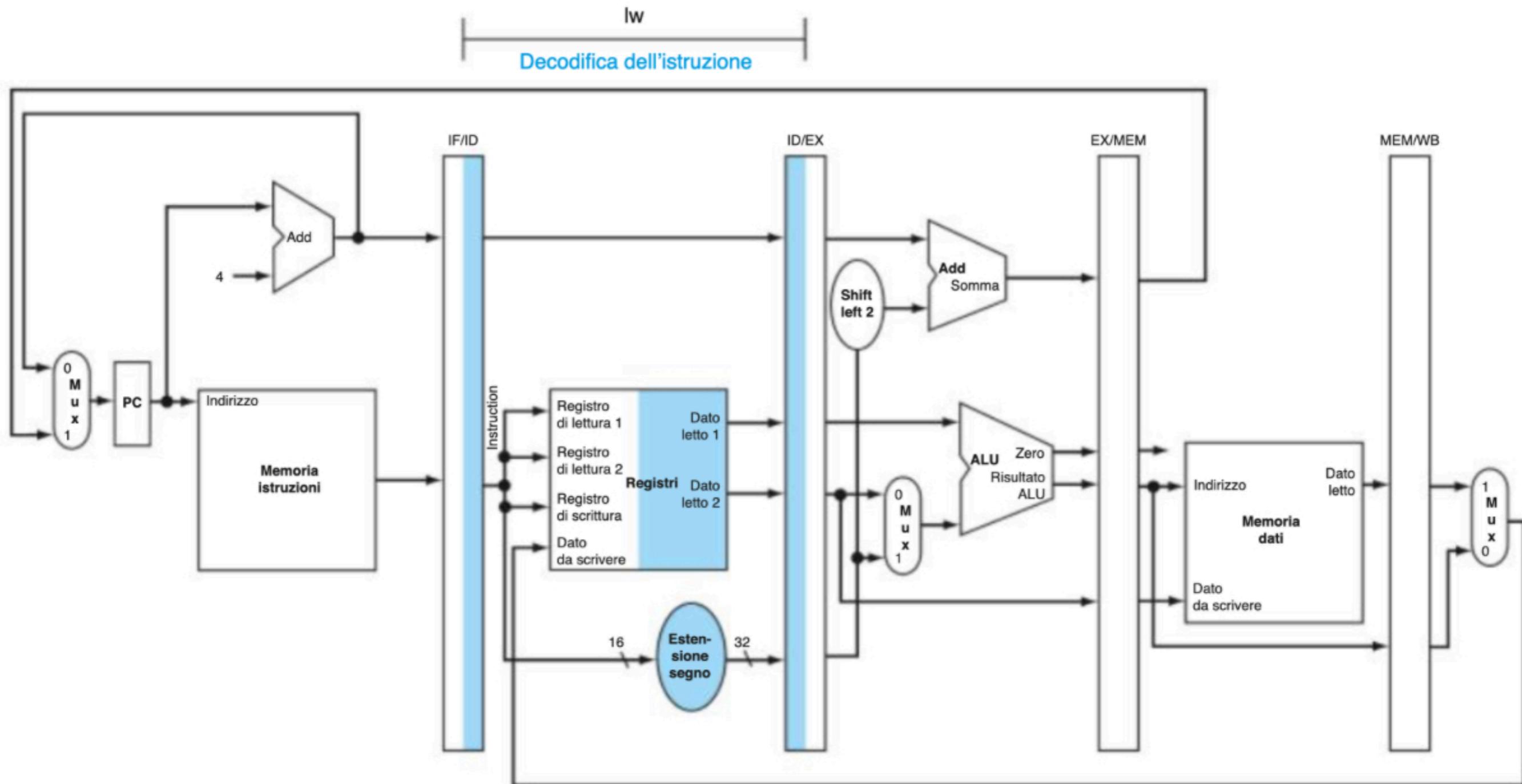
# Funzionamento della Pipeline

- Flusso di istruzioni ciclo per ciclo attraverso l'unità di elaborazione in pipeline
  - Diagramma della pipeline "a singolo ciclo di clock"
    - Mostra l'uso della pipeline in un singolo ciclo
    - Evidenzia le risorse utilizzate
  - Diagramma della pipeline "a multiplo ciclo di clock"
    - Grafico dell'operazione nel tempo
- Studieremo i diagramma "a singolo ciclo di clock" per la load e la store

# IF per Load, Store, ...

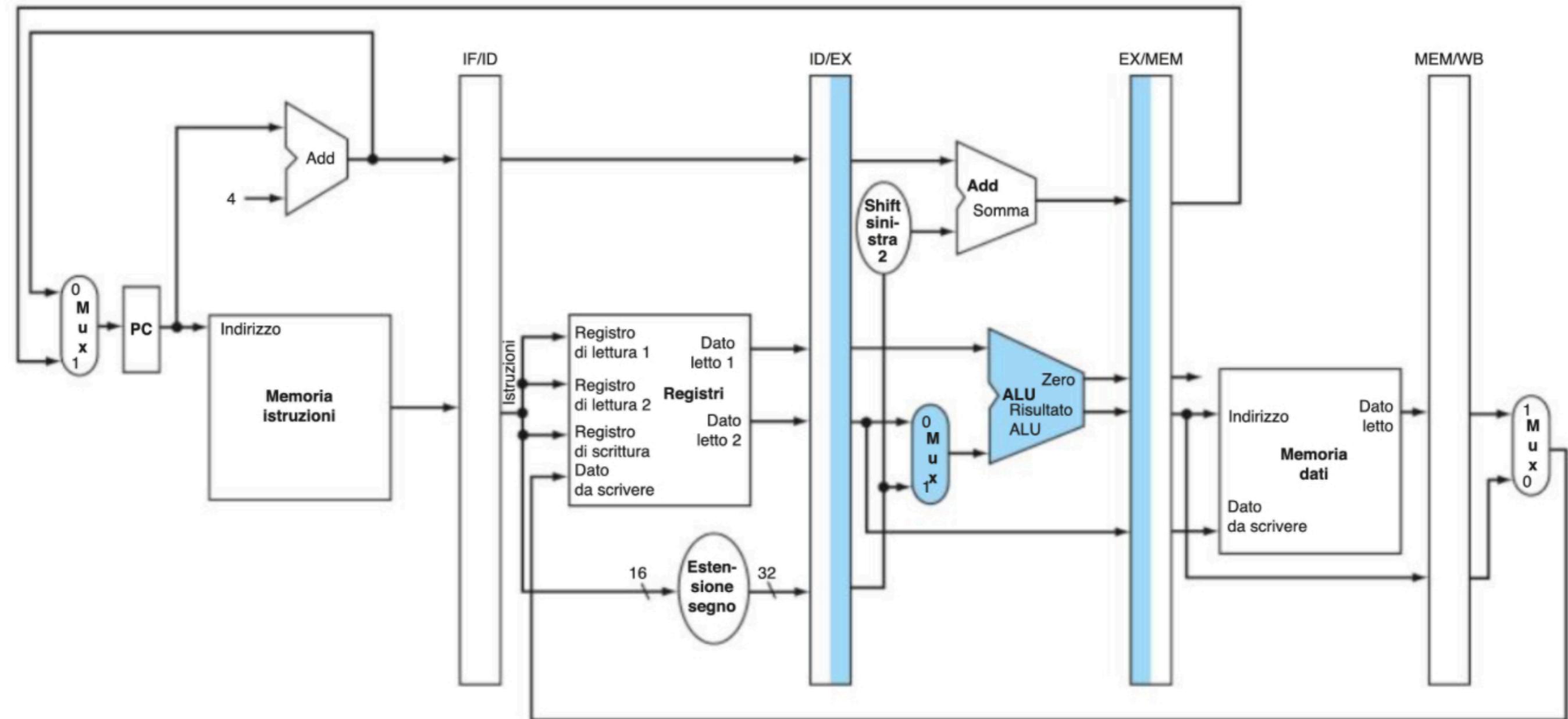


# ID per Load, Store, ...

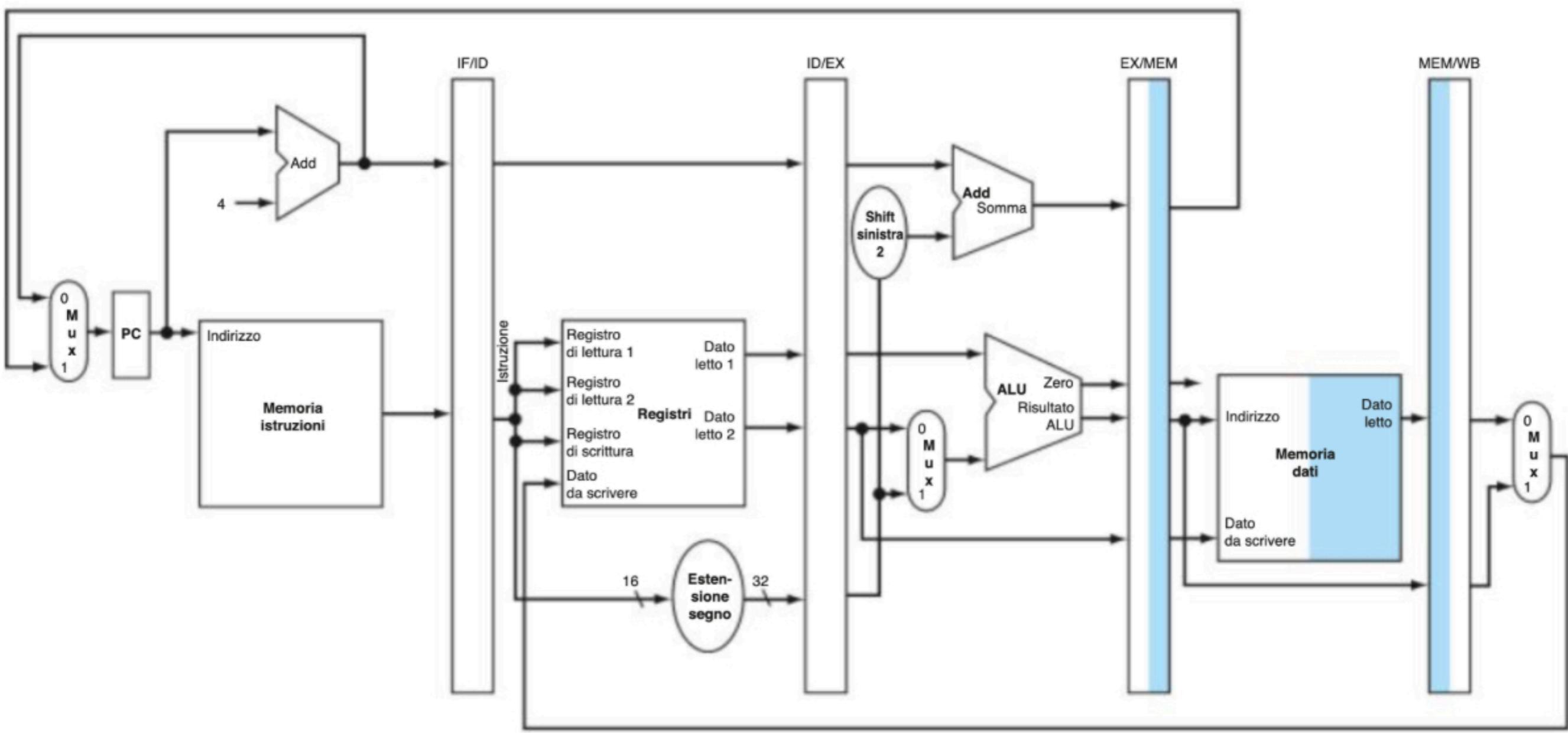


# EX per Load

$T_W$   
Esecuzione dell'istruzione



# MEM per Load

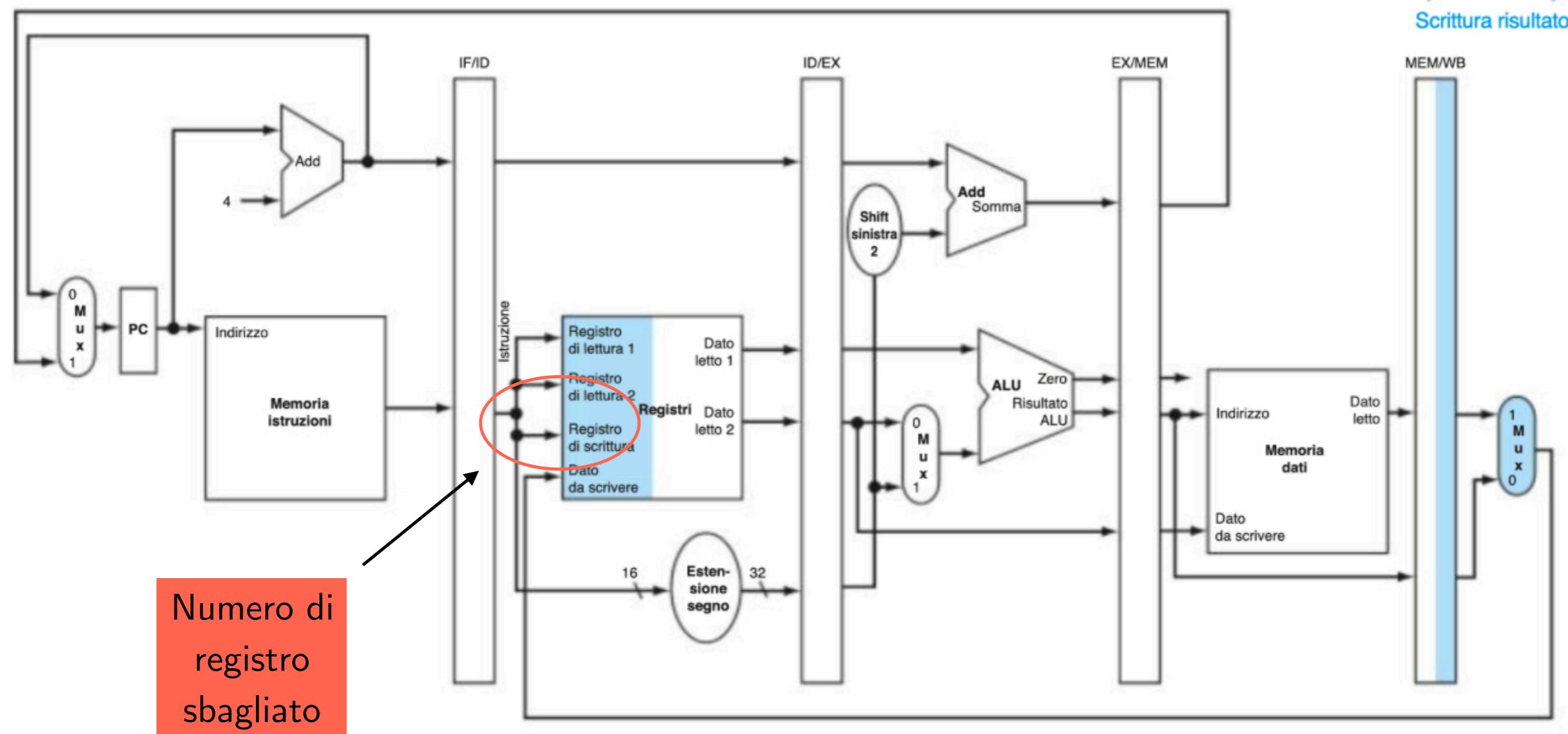


# WB per Load

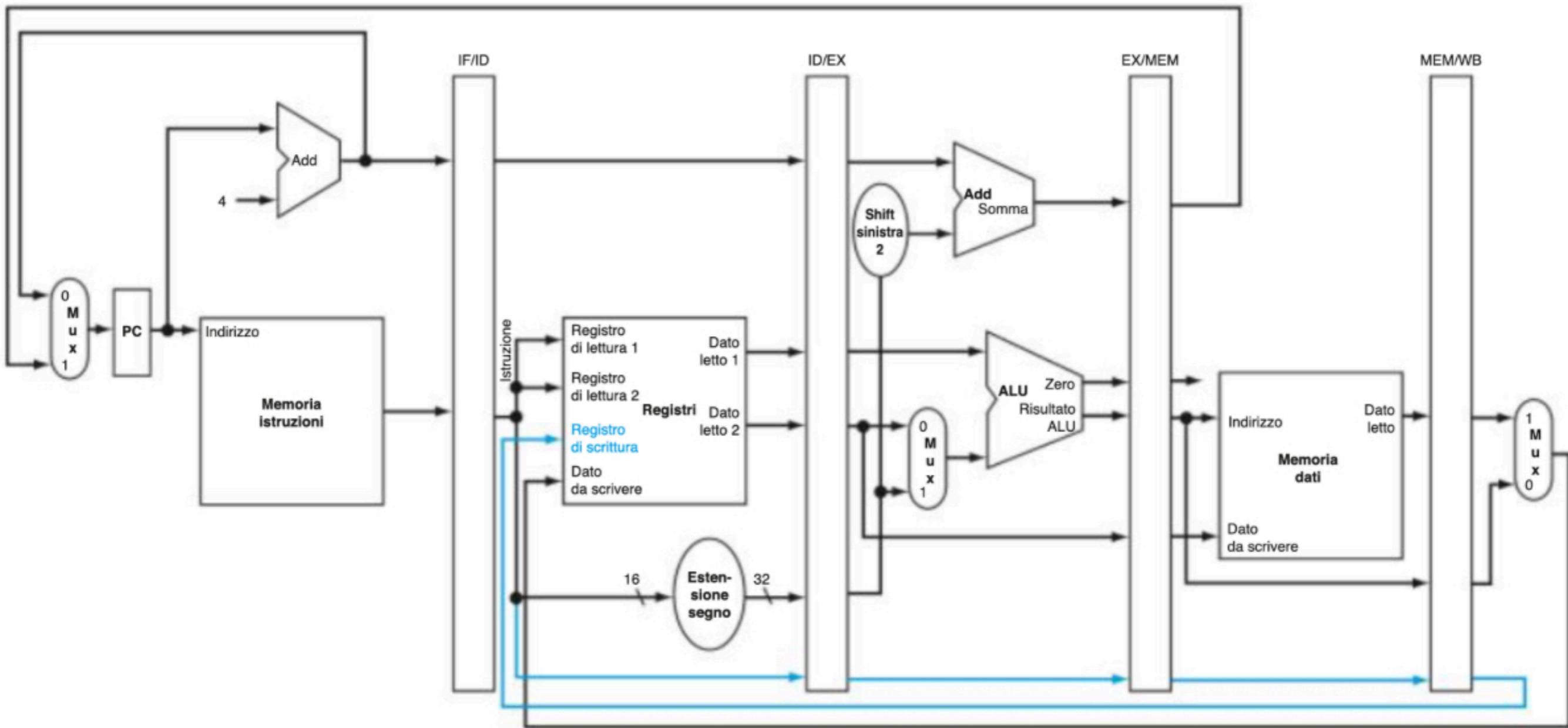
1w

Scrittura risultato

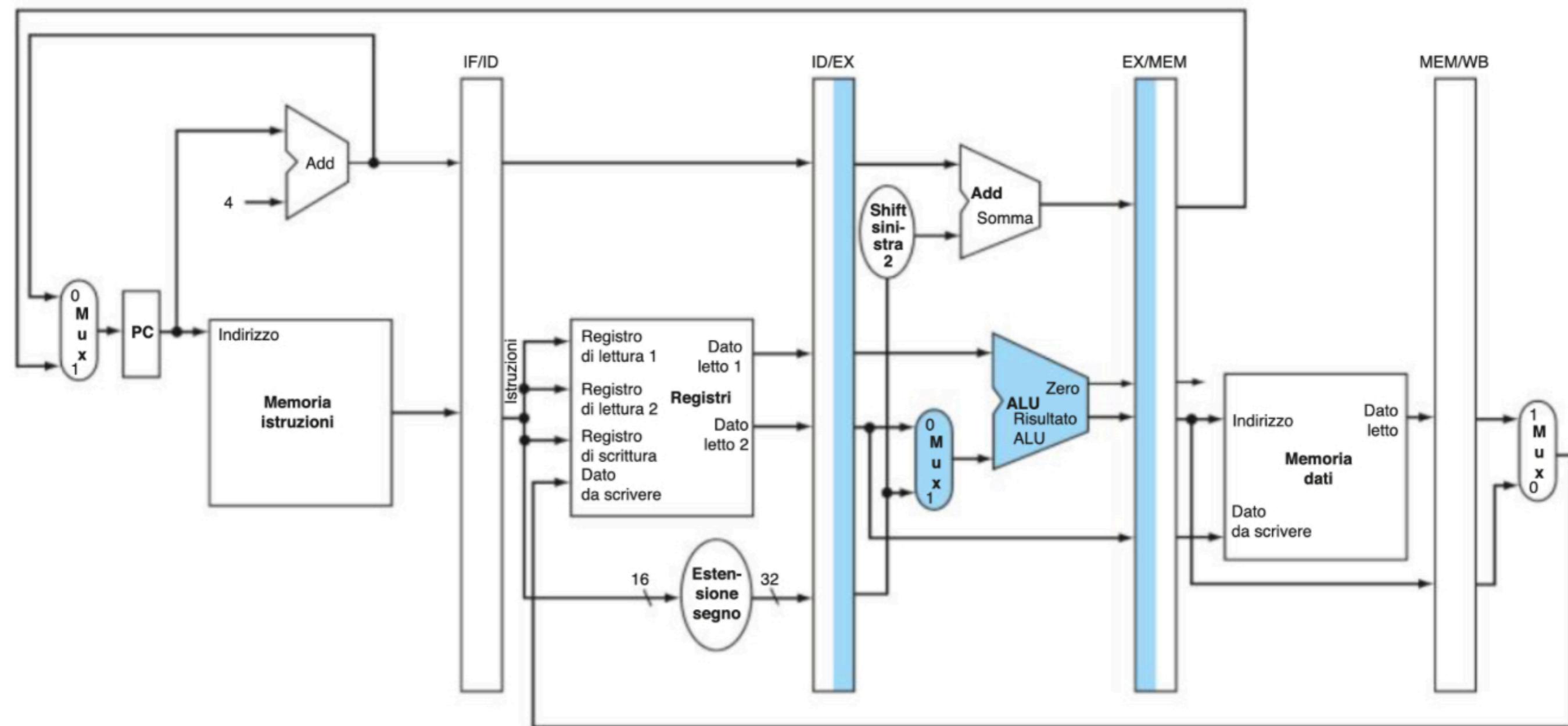
MEM/WB



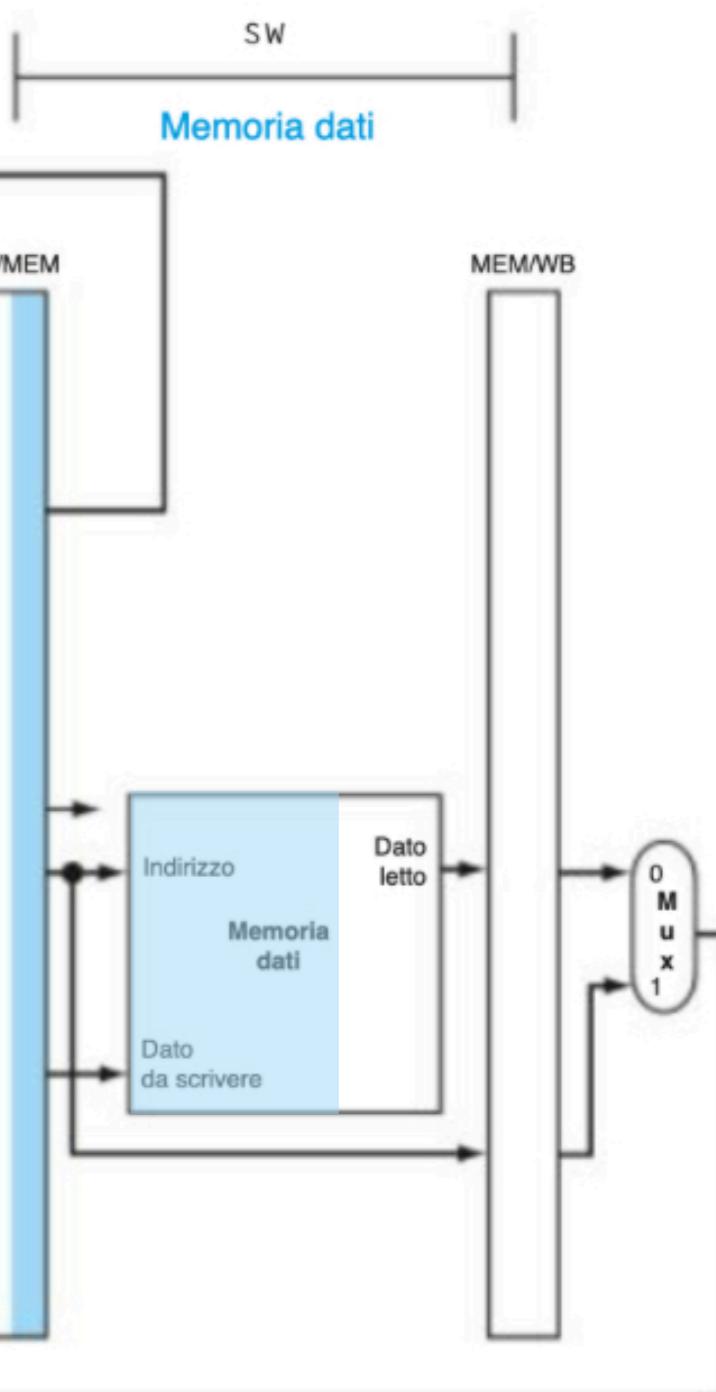
# Datapath corretto per Load



# EX per Store



# MEM per Store

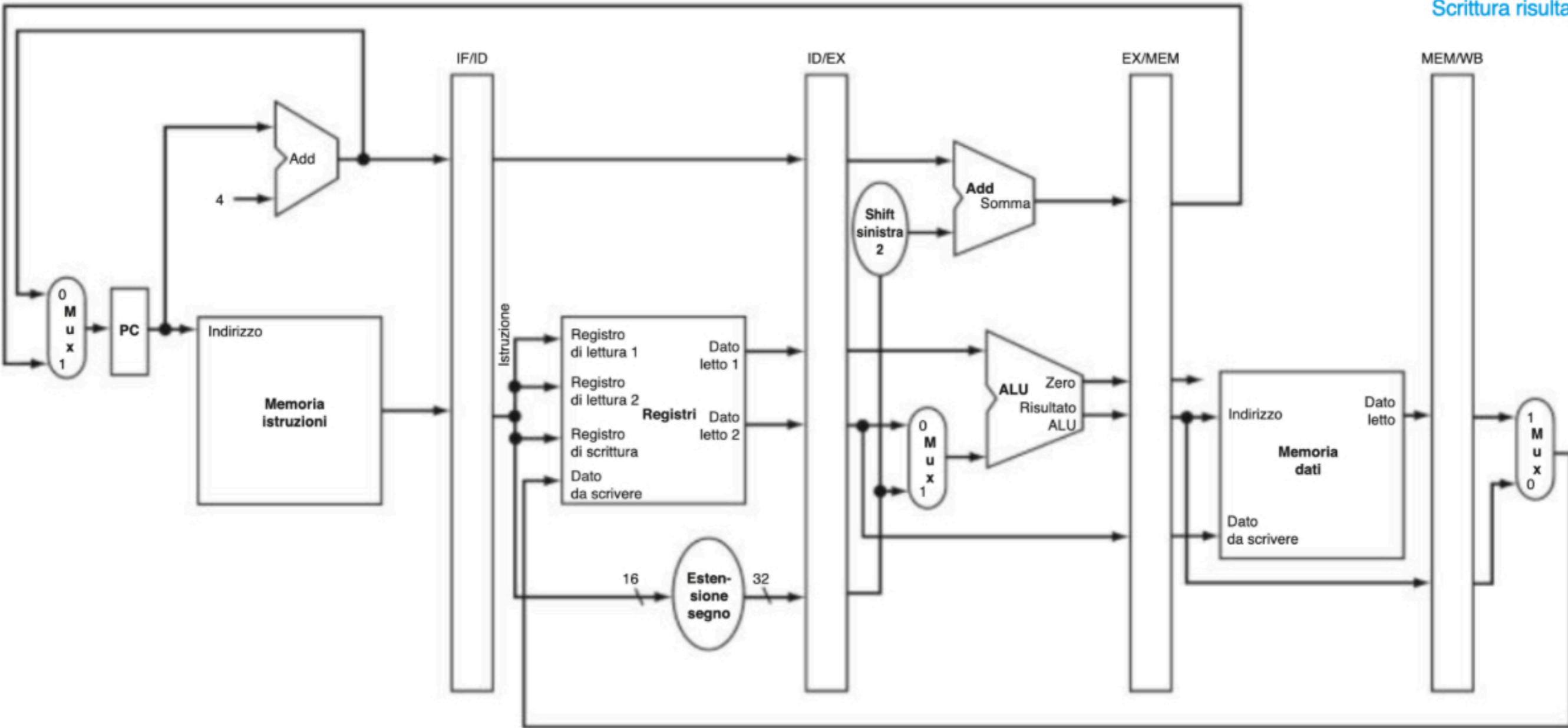


# WB per Store

SW

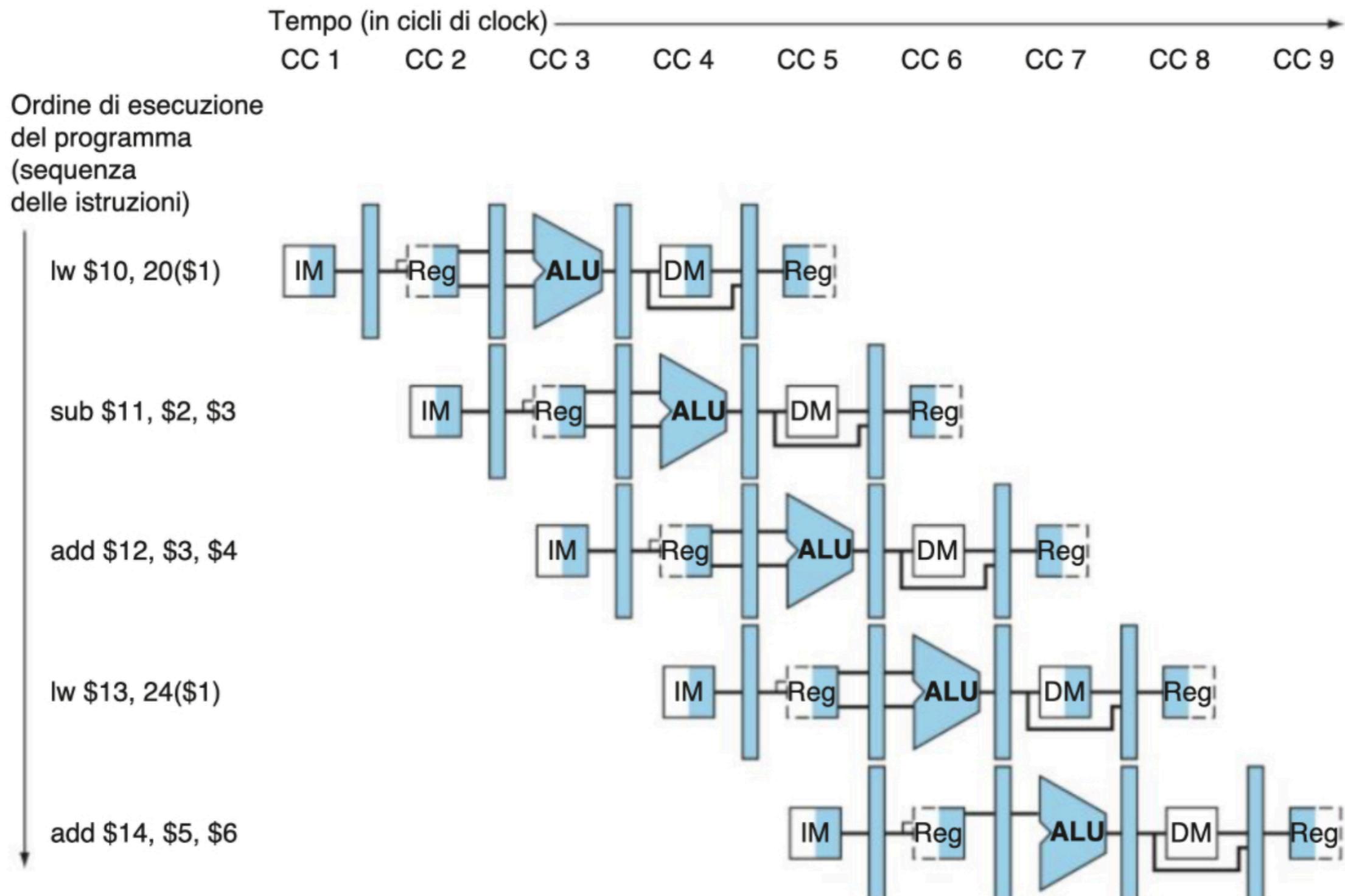
Scrittura risultato

MEM/WB



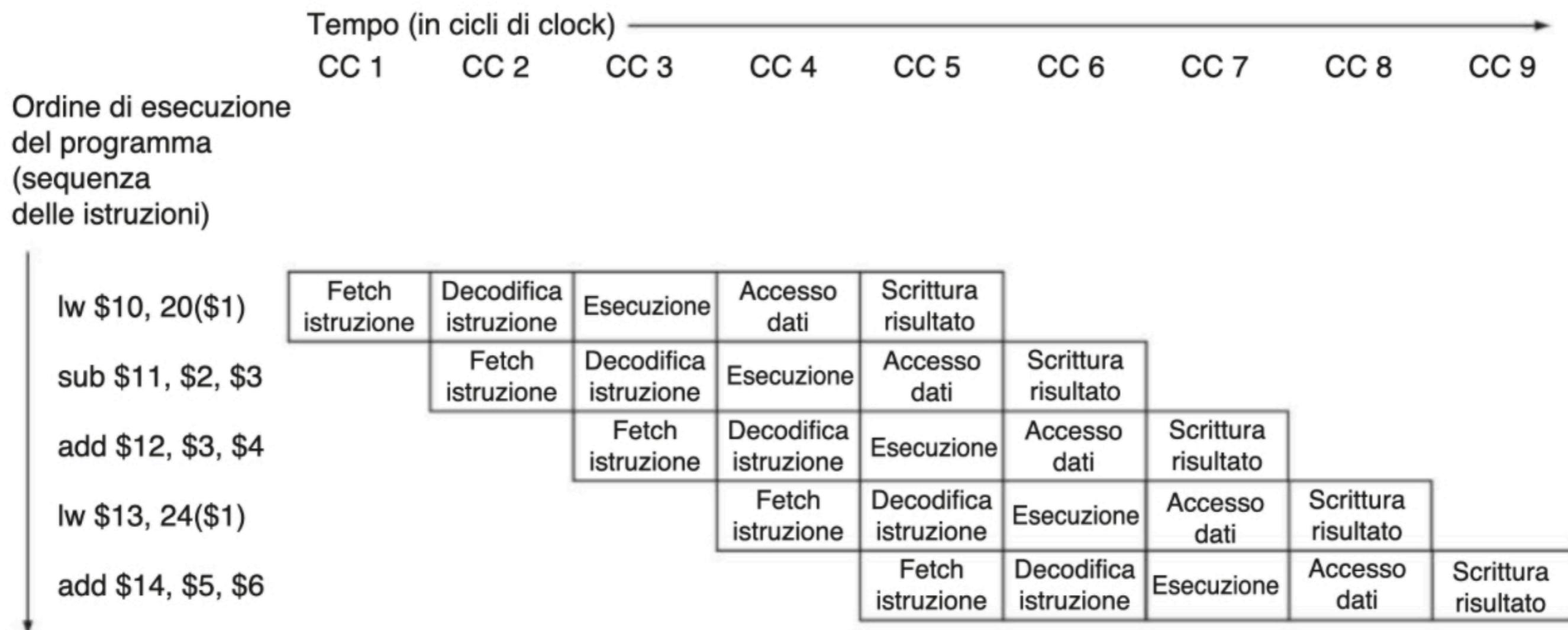
# Diagramma pipeline multi-ciclo

- Forma che mostra l'uso delle risorse



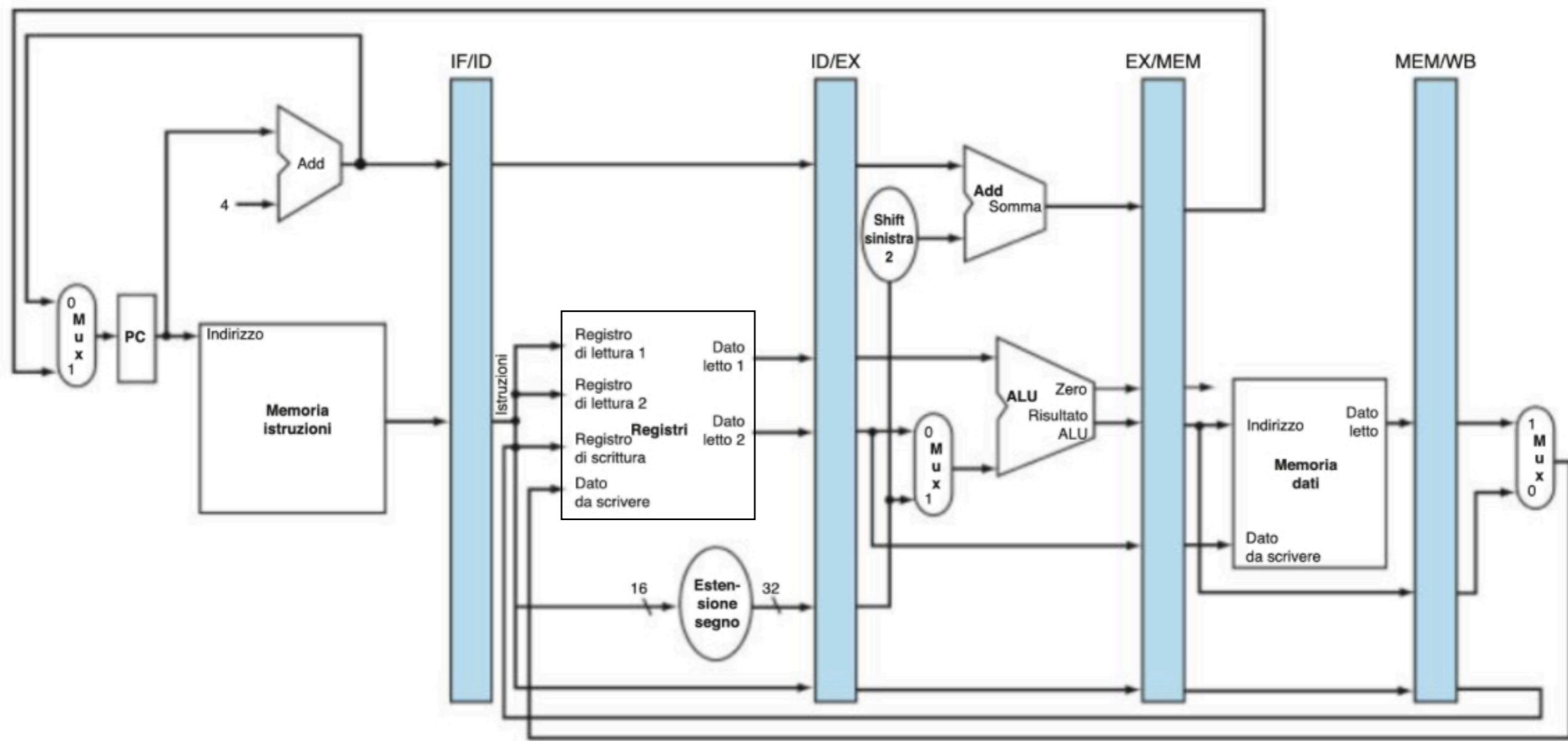
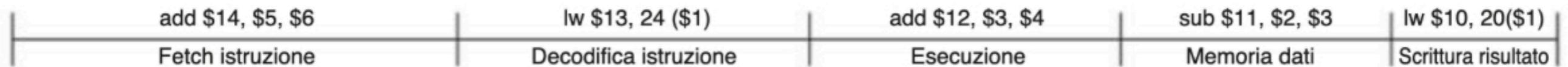
# Diagramma pipeline multi-ciclo

- Forma tradizionale

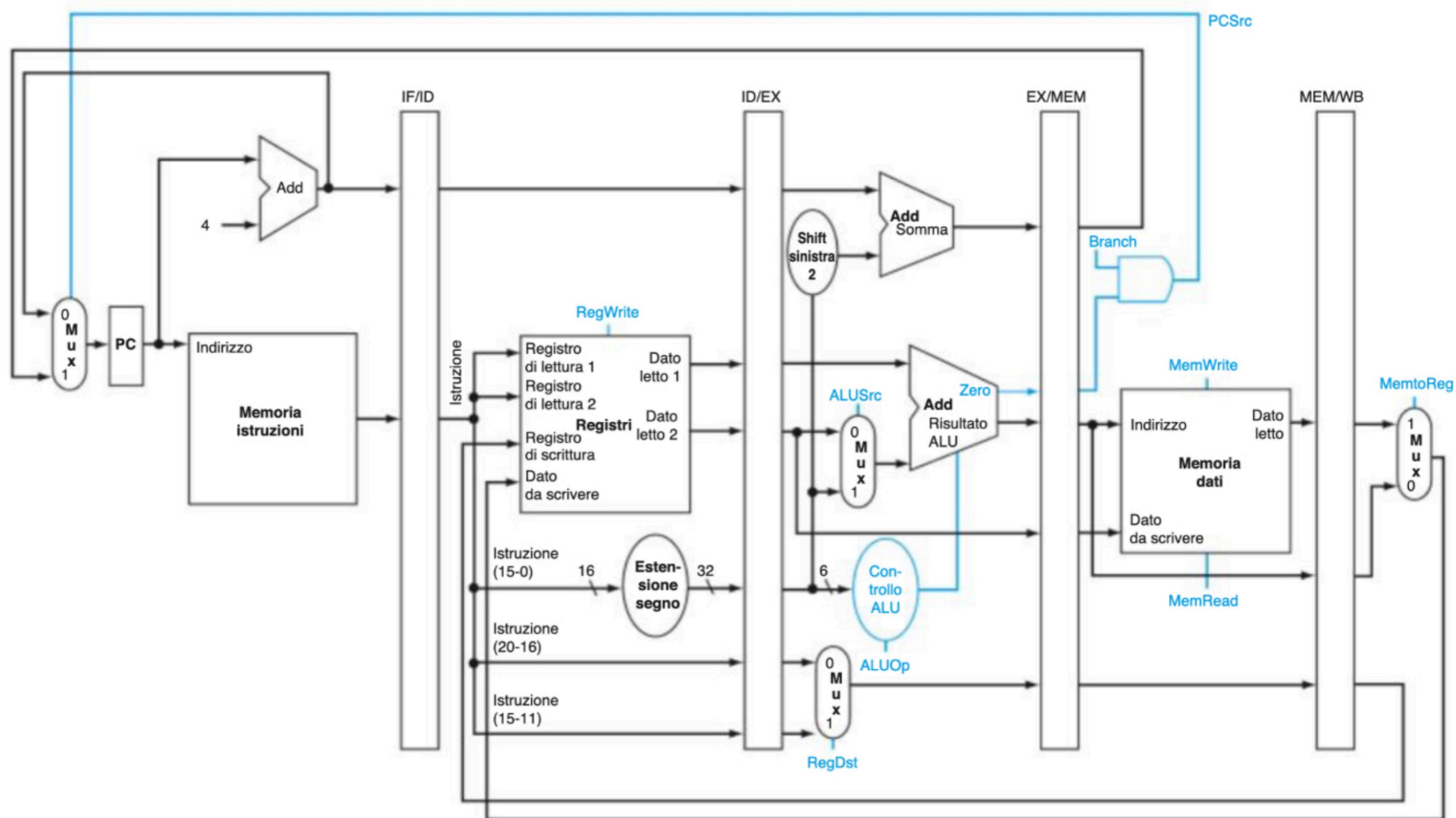


# Diagramma pipeline a ciclo singolo

- Stato della pipeline in un dato ciclo

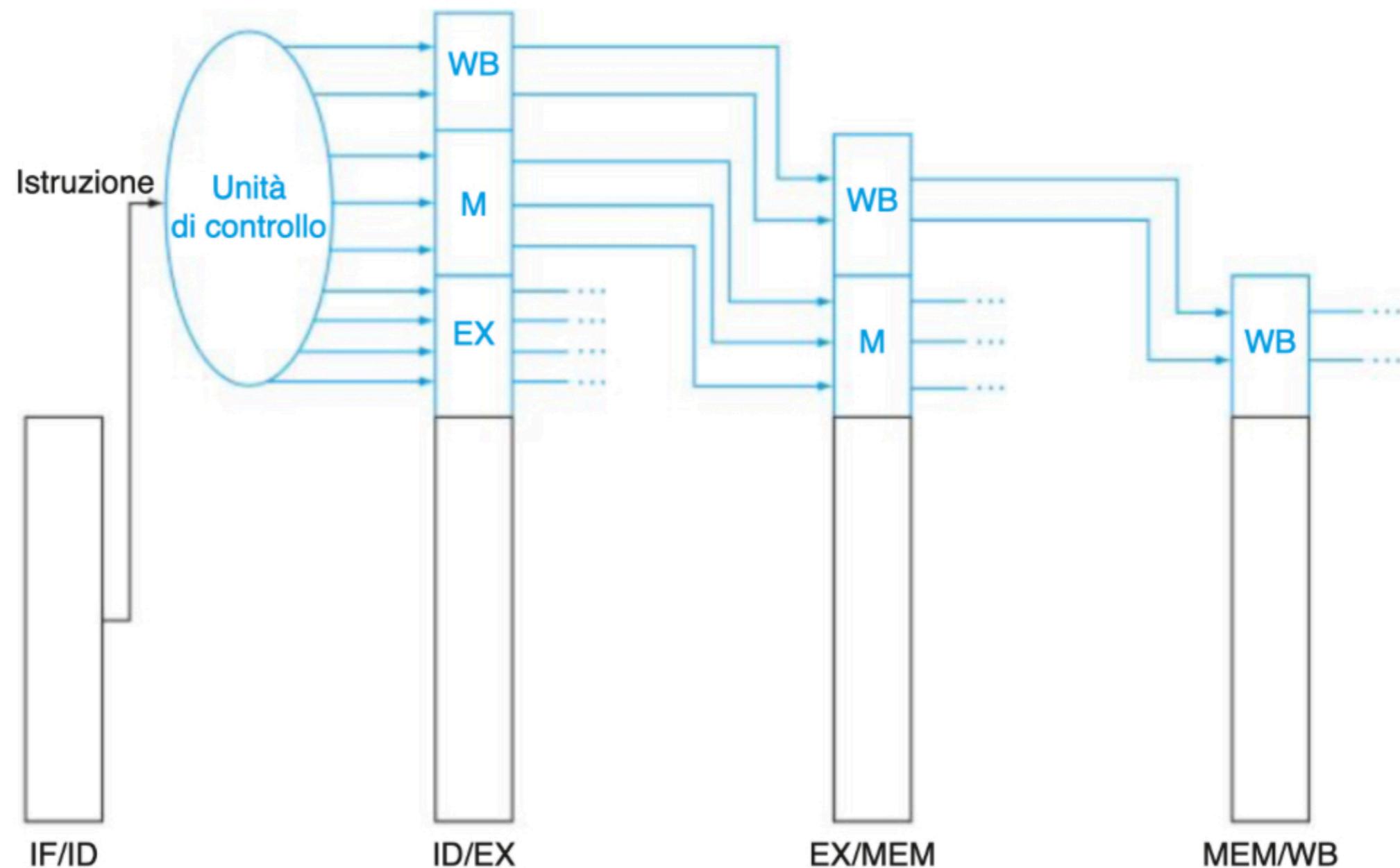


# Controllo della pipeline (semplificato)

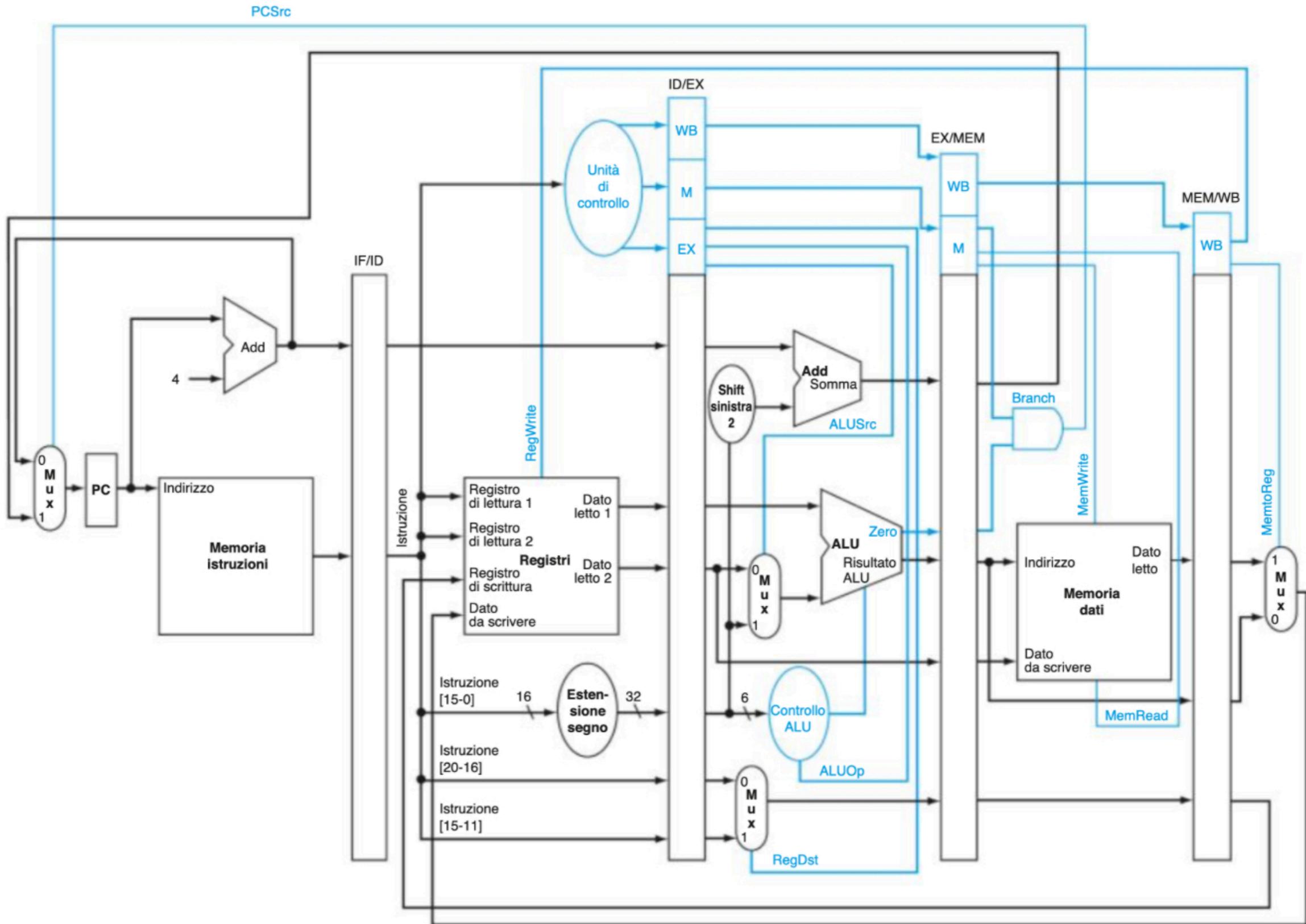


# Controllo della pipeline

- I segnali di controllo sono derivati dall'istruzione
  - Come nell'implementazione a ciclo singolo



# Controllo della pipeline

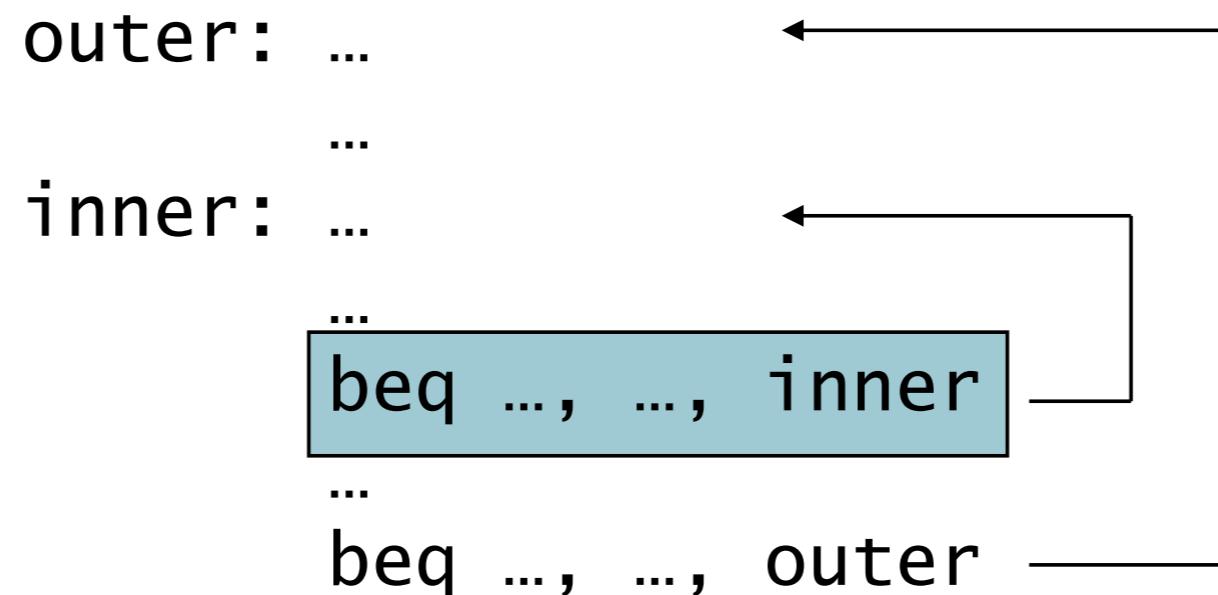


# Predittore dinamico dei salti

- Nelle pipeline lunghe e superscalari, la penalità di salto è molto significativa
- Si usa la predizione dinamica
  - Buffer di predizione dei salti (aka tabella di storia dei salti)
  - Indicizzato dagli indirizzi delle istruzioni di salto più recenti
  - Memorizza i risultati (preso/non preso)
  - Per eseguire un salto
    - Controllare la tabella, aspettandosi lo stesso risultato
    - Iniziare il fetch con l'istruzione corrispondente al risultato predetto
    - Se la predizione risulta sbagliata, svuotare la pipeline e invertire la predizione

# Predittore a 1 bit: limitazioni

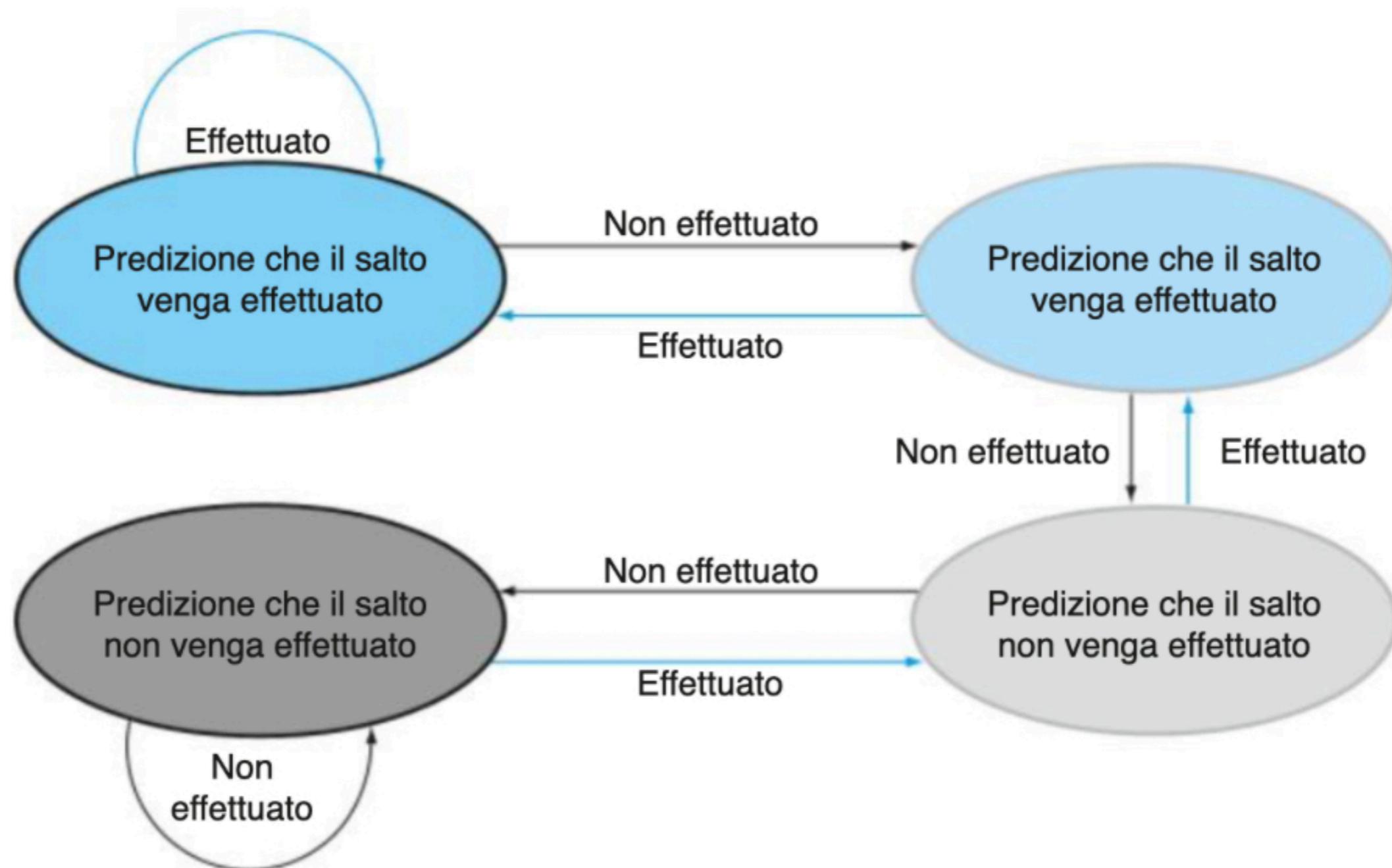
- I salti nei cicli interni sono predetti male due volte



- La predizione sbagliata "preso" avviene all'ultima iterazione del ciclo interno
- La predizione sbagliata "non preso" avviene alla prima iterazione del ciclo interno nella successiva iterazione del ciclo esterno

# Predittore a 2 bit

- Si cambia la predizione solo dopo due predizioni sbagliate consecutive



# Calcolare la destinazione di salto

- Anche col predittore, è sempre necessario calcolare l'indirizzo di destinazione
  - Penalità di 1 ciclo per un salto "preso"
- Buffer di predizione dei salti
  - Cache degli indirizzi di destinazione
  - Indicizzato da PC quando l'istruzione è prelevata
    - In caso di hit e di predizione di salto "preso", si può prelevare subito l'istruzione di destinazione

# Eccezioni e interruzioni

- Eventi "inattesi" richiedono un cambiamento nel flusso di controllo
  - ISA diverse usano termini diversi
- Eccezione
  - Sollevata all'interno della CPU
    - Per esempio, opcode non definito, overflow, chiamata di sistema
- Interruzione
  - Proviene da un controllore I/O esterno
  - Gestirle senza sacrificare le prestazioni è difficile

# Gestione delle eccezioni

- Nel MIPS le eccezioni sono gestite da coprocessore di controllo di sistema (CP0)
- Salvare PC dell'istruzione che ha sollevato l'eccezione (o interrotta)
  - Nel MIPS: program counter delle eccezioni (EPC)
- Salvare la causa del problema
  - Nel MIPS: registro causa
  - Assumeremo sia di 1 bit
    - 0 per opcode non definito, 1 per overflow
- Salto all'handler all'indirizzo 8000 0180

# Un meccanismo alternativo

- Interrupt vettorizzati
  - L'indirizzo dell'handler è determinato dalla causa
- Esempio:
  - opcode non definito: C000 0000
  - overflow: C000 0020
  - ....: C000 0040
- Le istruzioni possono:
  - Gestire l'interruzione
  - Saltare all'handler vero e proprio

# Azioni dell'handler

- Leggere la causa e trasferire il controllo all'handler competente
- Determinare l'azione necessaria
- Se recuperabile
  - Eseguire le azioni correttive
  - Usare EPC per ritornare al programma
- Altrimenti
  - Terminare il programma
  - Riportare l'errore usano EPC, causa, ...

# Eccezioni nella pipeline

- Un'altra forma di hazard sul controllo
- Si consideri un overflow sulla somma nello stadio EX

add \$1, \$2, \$1

- Evitare che \$1 sia "sporcato"
- Completare le istruzioni precedenti
- Scartare la add e le istruzioni successive
- Impostare i valori dei registri EPC e causa
- Trasferire il controllo all'handler
- Simile alla predizione sbagliata dei salti
- Usa molto dello stesso hardware

# Proprietà di un'eccezione

- Eccezioni recuperabili
  - La pipeline può scartare l'istruzione
  - L'handler va in esecuzione, quindi ritorna all'istruzione
    - Ri-prelevata ed eseguita da capo
- PC è salvato nel registro EPC
  - Identifica l'istruzione che ha causato l'eccezione
  - In realtà, si memorizza PC + 4
    - L'handler deve aggiustare il valore

# Esempio di eccezione

- Eccezione sulla add in

```
40  sub  $11, $2, $4  
44  and  $12, $2, $5  
48  or   $13, $2, $6  
4C  add  $1,  $2, $1  
50  slt   $15, $6, $7  
54  lw    $16, 50($7)
```

...

- Gestore

```
80000180  sw   $25, 1000($0)  
80000184  sw   $26, 1004($0)
```

...

# Eccezioni multiple

- Il pipelining sovrappone molteplici istruzioni
  - Si possono verificare molteplici eccezioni in una volta
- Approccio semplice: gestire l'eccezione della prima istruzione
  - Scartare le istruzioni successive
  - Eccezioni "precise"
- Nelle pipeline complesse
  - Molteplici istruzioni per ciclo
  - Completamento fuori ordine
  - Mantenere eccezioni precise è difficile!

# Eccezioni imprecise

- Semplicemente si blocca la pipeline e si salva lo stato
  - Inclusa/e la/e causa/e dell'eccezione
- Si lascia lavorare l'handler
  - Quale/i istruzione/i ha/hanno causato le eccezioni
  - Quali completare e quali scartare
    - Può richiedere un completamento "manuale"
- Semplifica l'hardware, ma rende più complesso il software dell'handler
- Non fattibile per pipeline complesse con esecuzione parallela fuori ordine

# Cortex A8 e Intel i7

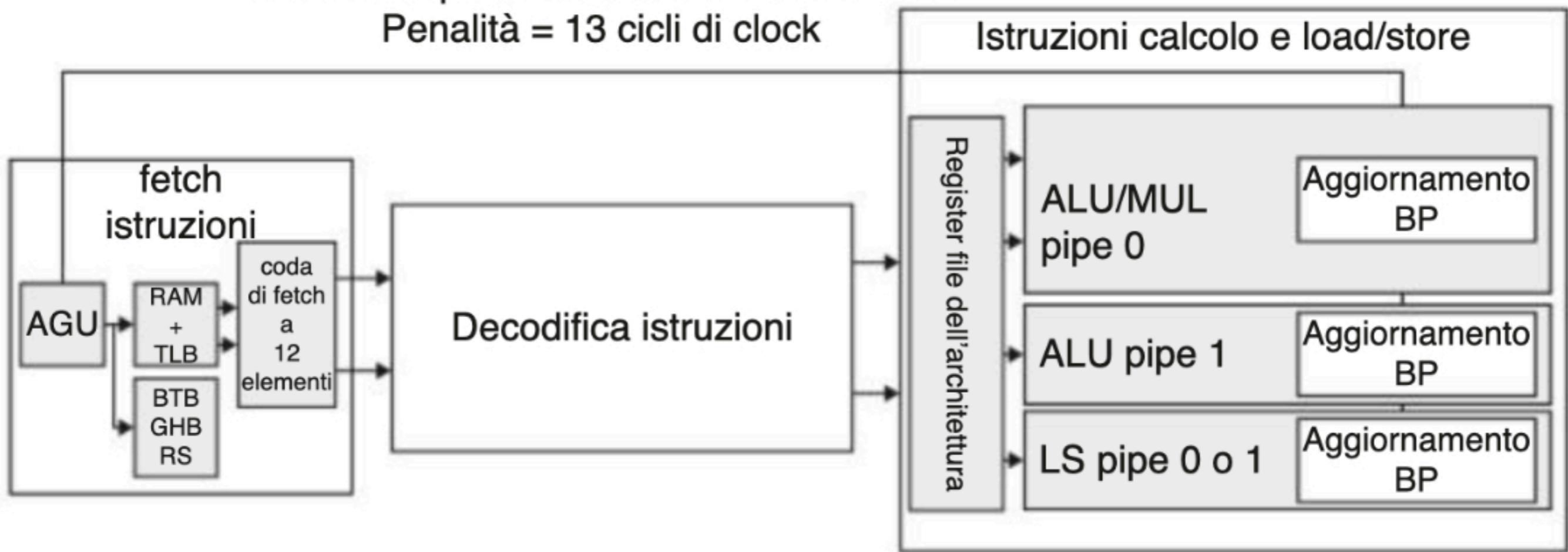
Processore	ARM A8	Intel Core i7 920
Mercato	Dispositivi Mobili Personali	Server, Cloud
Potenza termica di progetto	2 watt	130 watt
Frequenza di clock	1 GHz	2,66 GHz
Core/chip	1	4
Virgola mobile?	No	Sì
Parallelizzazione dell'esecuzione?	Dinamica	Dinamica
Istruzioni/ciclo di clock (picco)	2	4
Stadi della pipeline	14	14
Scheduling della pipeline	Statico in ordine	Dinamico fuori ordine, con speculazione
Predizione dei salti	a 2 livelli	a 2 livelli
Cache 1° livello/core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
Cache 2° livello/core	128-1024 KiB	256 KiB
Cache 3° livello (codivisa)	–	2-8 MiB

# Arm Cortex A8 Pipeline

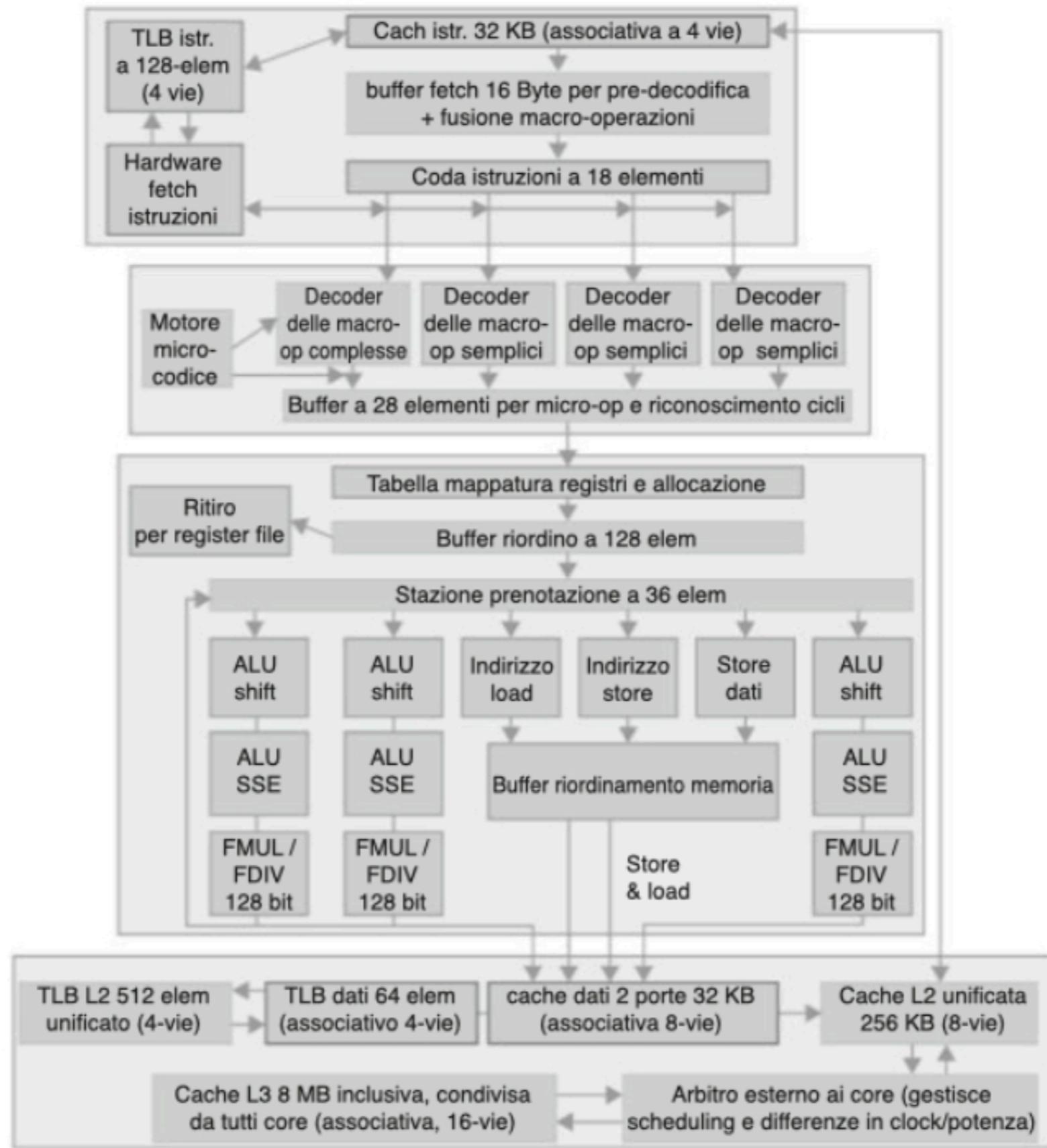
F0 F1 F2 D0 D1 D2 D3 D4 E0 E1 E2 E3 E4 E5

Errore nella predizione dei salti condizionati

Penalità = 13 cicli di clock



# Core i7 Pipeline



# Errori

- Il pipelining è semplice (!)
  - L'idea di base è semplice
  - Il diavolo sta nei dettagli
    - Per esempio, identificare gli hazard sui dati
- Il pipelining è indipendente dalla tecnologia
  - Allora perché non abbiamo sempre usato il pipelining?
  - Più transistor permettono tecniche più avanzate
  - La progettazione di ISA con pipeline deve tenere conto dei trend tecnologici

# Trabocchetti

- Una progettazione scadente dell'ISA può rendere il pipelining più difficile
  - Per esempio, insiemi di istruzioni complesse (VAX, IA-32)
    - Overhead significativo per far funzionare il pipelining
    - Approccio microistruzioni IA-32
  - Per esempio, modi di indirizzamento complessi
    - Indirezione della memoria, effetti collaterali sull'aggiornamento dei registri
  - Per esempio, salti ritardati
    - Le pipeline avanzate hanno lunghi slot di ritardo

# Note conclusive

- L'ISA influenza il progetto dell'unità di elaborazione e del controllo
- L'unità di elaborazione e il controllo influenzano il progetto dell'ISA
- Il pipelining migliora il throughput delle istruzioni usando il parallelismo
- Più istruzioni completate al secondo
- La latenza di ogni istruzione non è ridotta
- Hazard: strutturali, sui dati, sul controllo